gui

October 30, 2024

[81]:

```
# Import libraries
import os
import cv2
import glob
import numpy as np
from tkinter import *
from PIL import Image, ImageDraw, ImageGrab
from keras.models import load_model
```

[82]:

```
# Load the model
model = load_model(r'C:\Users\Rohit\OneDrive\Desktop\ROHIT\jupyter\model.h5')
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
  ↪metrics=['accuracy'])
print("Model loaded successfully. Ready for predictions.")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
```

```
Model loaded successfully. Ready for predictions.
```

[83]:

```
# Create a main window (named as root)
root = Tk()
root.resizable(0, 0)
root.title("Handwritten Digit Recognition GUI App")
```

[83]:

```
''
```

[84]:

```
# Initialize few variables
lastx, lasty = None, None
image_number = 0
```

[85]:

```
# Create a canvas for drawing
cv = Canvas(root, width=640, height=480, bg='white')
cv.grid(row=0, column=0, pady=2, sticky=W, columnspan=2)
```

[86]:

```
# Add buttons and labels
```

```
btn_recognize = Button(text="Recognize Digit", command=lambda: Recognize_Digit())
btn_recognize.grid(row=2, column=0, pady=1, padx=1)

btn_clear = Button(text="Clear", command=lambda: clear_widget())
btn_clear.grid(row=2, column=1, pady=1, padx=1)
```

[87]:

```
# Function to clear the canvas
def clear_widget():
    global cv
    cv.delete("all")
```

[88]:

```
# Function to handle mouse events
def activate_event(event):
    global lastx, lasty
    cv.bind('<B1-Motion>', draw_lines)
    lastx, lasty = event.x, event.y
```

[89]:

```
# Function to draw lines on the canvas
def draw_lines(event):
    global lastx, lasty
    x, y = event.x, event.y
    cv.create_line((lastx, lasty, x, y), width=8, fill='black', capstyle=ROUND, smooth=True, splinesteps=12)
    lastx, lasty = x, y
```

[90]:

```
def Recognize_Digit():
    global image_number
    predictions = []
    percentage = []
    filename = f'image_{image_number}.png'
    widget = cv

    # Get the widget coordinates
    x = root.winfo_rootx() + widget.winfo_x()
    y = root.winfo_rooty() + widget.winfo_y()
    x1 = x + widget.winfo_width()
    y1 = y + widget.winfo_height()

    # Grab the image and save it in PNG format
    ImageGrab.grab().crop((x, y, x1, y1)).save(filename)

    # Read the image in color format
    image = cv2.imread(filename, cv2.IMREAD_COLOR)

    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Applying Otsu thresholding
    ret, th = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.
↪THRESH_OTSU)

    # Find contours
    contours, _ = cv2.findContours(th, cv2.RETR_EXTERNAL, cv2.
↪CHAIN_APPROX_SIMPLE)
    for cnt in contours:
        # Get bounding box and extract ROI
        x, y, w, h = cv2.boundingRect(cnt)

        # Create rectangle around detected digit
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 1)

        top = int(0.05 * th.shape[0])
        bottom = top
        left = int(0.05 * th.shape[1])
        right = left

        # Extract the image ROI
        roi = th[y - top:y + h + bottom, x - left:x + w + right]
        if roi.size == 0:
            print("Empty ROI!")
            continue  # Skip to the next contour

        # Resize ROI image to 28x28 pixels
        img = cv2.resize(roi, (28, 28), interpolation=cv2.INTER_AREA)

        # Reshape the image to support model input
        img = img.reshape(1, 28, 28, 1)

        # Normalize the image to support model input
        img = img / 255.0

        # Predict the result
        try:
            pred = model.predict([img])[0]
            final_pred = np.argmax(pred)
            confidence = int(max(pred) * 100)
            data = f"{final_pred} {confidence}%"

            # Draw predicted result on the image
            cv2.putText(image, data, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,␣
↪(255, 0, 0), 1)
        except Exception as e:
            print("Error during prediction:", str(e))
    # Show the predicted results in a new window
    cv2.imshow('Predicted Image', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

[91]:

```
# Bind the activate event for drawing
cv.bind('<Button-1>', activate_event)

# Main loop
root.mainloop()
```

```
1/1               0s 85ms/step
1/1               0s 28ms/step
1/1               0s 28ms/step
1/1               0s 26ms/step
1/1               0s 30ms/step
1/1               0s 26ms/step
1/1               0s 32ms/step
```

[ ]:

```

```